Toolbox Manual

for the

Window Manager

Toolbox Manual For The Window Manager

by Saul Greenberg

Preface to the June 30th Edition

Man/machine interfaces to windowed systems are difficult to set up for two reasons. First, they are complex to build, for the flexibility they offer the user must be explicitly catered to by the programmer. Second, it is difficult to present an interface that is consistent between applications, especially when there are many people designing them. This toolbox attempts to remedy these problems by providing high level facilities for window manipulation. These minimize programming and encourage interface consistency. At the same time, low level primitives are available for complete programming flexibility.

To use this manual, go to the Appendix and scan through the packages available. Be aware of the high level functions, otherwise you will find yourself re-inventing the wheel with the primitives. At the end of each section, you will find example programs illustrating the use of the new concepts. Execute these programs before writing any code, to see what simple tasks can be done. Afterwards, look at the program code. The tri_message program in Section B is a particularly good illustration of a simple interface. When ready, you can copy these program directly and use them as a template for your own needs.

You need only minimal knowledge of the Jade system to create a windowed system. You must know how to initialize your processes, manipulate vterm, and get the window manager variables — see the example programs for this. Although you should understand what is going on, you can probably get away with copying the relevant pieces of example program segments.

The toolbox is open ended. If you write a module which can ease the programming burdens of others, we will gladly add it to the package.

One last thing. In the (very unlikely) event that you spot errors in the manual or bugs in the code, please report them to Saul Greenberg (rm 281 Maths building) or send electronic mail to greenberg@vaxb.

TABLE OF CONTENTS

A.

Primitive Window Calls

Revision of 30 June 1985

A. PRIMITIVE WINDOW CALLS

This section describes a modest extension of all the window manager calls present within Jade. Its primary purpose is to offer a clean C-style interface to the window manager and some enhancements not normally available through the equivalent Jade calls. It offers the following capabilities:

- A *C-style syntax* to Jade window requests

- *Automatic maintanance of its own jipc buffer* -- this package will not interfere with other jipc calls.

- *Events are queued one-deep*, allowing events to be put back. This essentially provides a look-ahead capability.

- *Buffering of output requests* - output requests may be sent immediately (unbuffered mode) or buffered and sent automatically and much more quickly.

- *Requests of system states* such as whether the buffer is in buffered or unbuffered mode, the status of the point stroke state (set or unset), etc.

This section catagorizes and describes all the window manager routines available to the applications programmer. With one exception, all parameters are described within the subsection introduction or the subroutine description.

The exception is *window*, a parameter found in virtually all routines but one. This parameter is of the type WINDOW and is returned by the window initialization routine (WInit). The subsequent discussions assume that the parameter window is declared as WINDOW.

NOTE: If this package is used, the application programmer should avoid using the jipc-style window manager messages described in the JADE manual. Also, scan through the rest of the manual before programming, as further sections describe high-level packages which may minimize your programming at the primitive level.

A.1. Initializing And Ending A Window

Before using this package, the system must be initialized by *w_init*, which returns a value used by all subsequent window manager calls. Initialization creates a window structure which keeps track of the current buffer, queues events and is aware of the current system state.

When the application no longer has need of the window, it should be ended by *w_end*, a housekeeping routine which frees up the window structure.

```
WINDOW w_init (window_manager, window_identifier)
   j_process_id window_manager;
   int window_identifier;
```

This function must be used before any other calls in the toolbox. A WINDOW pointer is returned from the call which is necessary for all subsequent window requests. It is your responsibility to supply the correct *window manager* and *identifier* to the window.

w_end (window)

End the window. This is normally used when the application has no further use for the window.

A.2. Requests Of Window Status

It is possible for the application to find out certain attributes of a window. These include finding the value of the window manager and identifier and the current state of the window buffering and the stroke mode. Also, one can find the size and color of a window.

j_process_id w_get_manager (window)

Return the window manager of the window.

int w_get_id (window)

Return the window identifier of the window.

int w_is_buffered (window)

Returns TRUE if the window is in cooked mode, else FALSE.

int w_is_stroke_mode (window)

Returns TRUE if the window is in stroke mode, else FALSE.

```
w_get_info (window, x1, y1, x2, y2, color)
  WINDOW window;
  int *x1, *x2, *y1, *y2, *color;
```

Return the absolute screen coordinates of the window and its current background color. The coordinates returned are the lower-left corner *(x1, y1)*, lower-right corner *(x2, y2)* and the background *color* (0 for black, 1 for white).

A.3. Buffering Output Requests

Output requests may be sent individually *(unbuffered mode)* or packaged together *(buffered mode)*. In unbuffered mode, every output request is sent immediately, with the consequence of relatively slow output to the screen. In buffered mode, output requests are automatically grouped together, with the consequence of relatively fast output to the screen. In buffered mode, there is normally no need to *flush* the output buffer as it is done automatically. In odd circumstances, a manual flush is possible.

When a window is initialized, it is put into *unbuffered mode* as a default.

w_buffered (window)

> Output requests are buffered and sent only when the buffer is full or when a non-output request is made.

w_unbuffered (window)

> All window manager requests are sent as individual messages. When a window is initialized, it is put into this state.

w_flush (window)

> Output requests are flushed immediately from the buffer.

A.4. Events

All user actions generate some type of input *event*, whether it be due to selecting a key on a keyboard, hitting a mouse button, or destroying a window. Event-based input is a fundamental difference between window-oriented and traditional stream input. It is important to understand and take advantage of this difference!

The following routines describes how events are detected, the types of events, and the parameters that can be retrieved when an event has occured. Finally, a small example is provided to illustrate event use.

Event detection.

The following routines provide the capability to see if any event has occurred, the retrieval of the event type, and the putting back of retrieved events.

int w_any_events (window)

> This reply returns TRUE if an event is waiting to be picked up by "w_get_next_event" and FALSE if no event is pending.

int w_get_next_event (window)

> Notifies the application that an event has occurred by returning the type of event (see ***Event types*** below). This will not return until an event has occured.

w_put_back_event (window)

> Puts back the last event. This provides a look-ahead facility which is not present in the standard system. An event can only be put back immediately after a "w_get_next_event" call. If a window manager request occurs between the two, an error will occur.

Event types.

There are only a limited number of input events that can occur. These events and the action that produced them are described below:

WM_MENU_SELECTION	The user has made a menu selection.
WM_POINT_DEPRESSED	The point button was depressed.
WM_POINT_RELEASED	The point button was released.
WM_POINT_STROKE	The point button was held down while the mouse was moved (see w_set_stroke_mode).
WM_SIZE_SET	The user has changed the window size.
WM_KEY	The user has hit a key on the keyboard.
WM_CANCEL	The user has depressed a menu button and released it without making a menu selection. This is interpretted as a cancel.
WM_DESTROY	The user has destroyed the window

Event parameters.

After an event has occured, some parameters may be retrieved by the application. For example, a WM_POINT_DEPRESSED event indicates that the application may retrieve the (x,y) coordinate of where the point press occured. It is important that these parameters are retrieved right after the event, for any other window request will destroy the information.

```
w_get_menu (window, pane, menu, item, new_window)
   int *pane, *new_window;
   char menu[12], item[12];
```

This call retrieves the parameters from a WM_MENU_SELECTION event. It indicates that the user has made a selection while in the returned *pane*. The name of the *menu* and *item* is returned. If a make window request is associated with the item, the id of the *new_window* will be returned, else a 0 (see w_make_window).

```
w_get_point (window, x_coord, y_coord)
   int *x_coord, *y_coord;
```

An (x,y) raster point coordinate can be retrieved after any one of the following events occur: WM_POINT_DEPRESSED, WM_POINT_RELEASED, WM_POINT_STROKE or WM_SIZE_SET.

```
w_get_key (window, *key)
   char key[50];
```

The returns the *key* typed by the user (a WM_KEY event). The string will contain one character for ordinary keys and up to fifty characters for COMMAND keys.

Example.

The following program segment illustrates how events may be used (parameters are assumed to be previously declared):

```
      ....
switch ( w_get_next_event (window) ) {
      case WM_KEY:
            w_get_key (window, key);
            ...
      case WM_POINT_DEPRESSED:
            w_get_point (window, &x_coord, &y_coord)
            ...
        ...
    }
```

A.5. Requests To The Window Manager

The window manager may be directed to do many tasks, such as creation and destruction of pop-up menus and help windows, changing and retrieving the cursor or title bar of the window, etc. The following routines direct the window manager to carry out the given request.

Although you should understand and try most of these routines, Section B of this manual provides simpler facilities (although slightly less flexible) for laying out title bars, pop-up menus and help windows and for changing the cursor description.

Titles.

The string in the title bar of the window may be changed or retrieved by the following routines:

```
w_set_title (window, window_title)
   char *window_title;
```

Replace the title bar of the window by the string *window_title.* The title may be truncated to fit.

```
w_get_title (window, window_title)
   char window_title[120];
```

Return the current title bar of the window in the string *window_title.*

Menus and help windows.

Pop-up menus and help windows and panes may be added, deleted or modified by the window manager.

```
w_set_pane_division (window, division_number, orientation, distance)
   int division_number;
   char orientation;
   int distance;
```

Sets up the pane division identified in ***division number*** to be either a vertical or horizontal ***orientation,*** with the division line a given ***distance*** from the lower left corner. Divisions are numbered starting with zero. There are sixteen pane divisions, initially all set at position zero. You may use the pre-defined definitions W_VERTICAL and W_HORIZONTAL for ***orientation.*** Panes are described in detail in Section III.B of the Jade user manual.

```
w_add_menu (window, division_vector, division_mask, menu_title, help)
   int division_vector;
   int division_mask;
   char *menu_title;
   char *help;
```

This call adds a new menu to the stack for a clump of panes. The *division_vector* and *division_mask* are the bit-vector and mask identifying the clump; giving a zero for the mask indicates that none of the division lines are to be looked at, so the menu applies to the whole window. *Menu_title* contains the title of the menu and the individual menu items. *Help* gives the contents of the associated help window. Both the *division_vector* and *division_mask* may be replaced by the constant W_IGNORE_PANE if the menu is to be applied to the whole window. A full description of menus is given in section III.B of the Jade user manual.

```
w_add_simple_menu (window, menu_title, help)
```

This macro is identical to the one above except that it ignores all references to panes.

```
w_delete_menu (window, menu_title)
   char *menu_title;
```

Delete the menu with the given *menu_title* from all panes where it occurs.

```
w_make_window (window, menu_title, menu_item, x_default, y_default)
   char *menu_title;
   char *menu_item;
   int x_default, y_default;
```

Directs the window manager to have the user create a window whenever the *menu_item* from the given *menu_title* is selected. The id of the new window will be made available to whoever sees the menu selection. The *default* size (in pixels) of the created window is specified by the two integers. A zero value for a default size will cause the window manager to select its own default for that dimension. Very small defaults are considered to be identical to zero.

```
w_add_help (window, division_vector, division_mask, help)
   int division_vector;
   int division_mask;
   char *help;
```

Adds a new help window with the contents *help* for the panes identified by the bit-vector and mask given in *division_vector* and *division_mask.* The new window will be on top of help windows added earlier and below any menu-related help windows. Both the *division_vector* and *division_mask* may be replaced by the constant W_IGNORE_PANE if the menu is to be applied to the whole window.

```
w_add_simple_help (window, help)
```

This macro is identical to the one above except that it ignores all references to panes.

```
w_delete_help (window, help_title)
   char *help_title;
```

Delete the help window called *help_title* from the stack from all panes.

```
w_item_help (window, menu_title, menu_item, help)
   char *menu_title;
   char *menu_item;
   char *help;
```

Associates a help window with contents *help* with a particular *menu_item* of a particular *menu_title*. The help window will be displayed only when that item is highlighted.

The pointing device cursor.

The current pointing device cursor may be changed or retrieved by the following requests. Note that the cursor description of any system cursor may be obtained by the Unix level command *showcus -d -s <cursor_name>*, which just has to be modified slightly for inclusion in these routines.

```
w_set_cursor (window, cursor_description, x_origin, y_origin)
   short        cursor_description[16];
   int x_origin;
   int y_origin;
```

Changes the pointing device cursor to the bit pattern given in *cursor_description* (the array should represent a 16X16 pixel pattern). The origin of the cursor (with respect to the upper left corner) is given by the x and y *origins* (range of zero to fifteen). The origin of a cursor is the point within the pattern which is "where the cursor is at". Note that this applies only to this window.

```
w_get_cursor (window, cursor_description, x_origin, y_origin)
   short        cursor_description[16];
   int x_origin;
   int y_origin;
```

Return a description of the current window cursor in *cursor_description*, and the (x,y) *origin* in the same form as the w_set_cursor routine.

Miscellaneous window requests.

```
w_set_stroke_mode (window, stroke_mode)
   int stroke_mode;
```

Sets whether the window manager should record the positions of the cursor during the time the point button is depressed or only the positions at the times of depression and release. Setting *stroke_mode* to TRUE means record it all; FALSE just the ends.

```
w_get_window_size (window, x_size, y_size)
   int *x_size, *y_size;
```

Gets the window size in pixels.

```
w_get_text_line (window, line_number, text_line, max_string_length)
   int line_number;
   char text_line[];
   int max_string_length;
```

Returns the character string representing the line of text on the given *line_number* of the window. The *max_string_length* of the text is returned is returned in *text_line* which must be large enough to accept it. The text returned is what the window manager believes to be there for stash/retrieve purposes. Any graphics requests (such as w_clear_rectangle or w_put_characters) will not affect the stashed text. It is thus possible to have a line of text returned which has no relationship to what is on the screen.

A.6. Output To The Window

The window manager may be directed to do many output tasks, such as: drawing characters at pixel or line/column positions; bit, vector and region graphics, etc. If you desire rapid output, you should probably put the window into cooked mode (see section A.3 Buffering output requests).

The coordinate system used for the character-oriented output operations is line one at the top of the screen, column one at the left edge. The coordinate system for graphics requests is x to the right, y up, with (0,0) in the lower left corner. If the window size is not an integral multiple of a character size, the characters are pushed to the bottom-left of the window, with the excess space on the right and the top.

Most of the routines described below contain the parameter *color* (an integer).The task of this parameter is to describe the color of the output as the background color, the contrast color and in some cases exclusive-or mode. Three definitions allow these to be described: W_BACKGROUND, W_CONTRAST and W_XOR. For example, a vector with color set to W_CONTRAST will draw white on a black screen and vice versa. Similarly, W_BACKGROUND would put a black line on a black screen and a white line on a white screen. Most output requests have macros which do away with the *color* parameter. Macros suffixed with '_b' do the operation in the background color, '_c' suffixes do it in the contrast color, while '_xor' draws in xor mode. For example, the following two requests are equivalent:

```
w_draw_string        (window, 1, 1, W_BACKGROUND, "Hello")
w_draw_string_b       (window, 1, 1, "Hello").
```

Window Graphics.

```
w_clear (window, color)
```

Clear the window to the given color. (Macros available - '_b', '_c' and '_xor'.)

Drawing text on the screen.

Text may be drawn on the screen using character-oriented output (line, column) or graphics-oriented output (x,y pixel position). They are not directly related - while stash and retrieve work on text graphics, they do not work with raster graphics.

```
w_draw_string (window, line, column, color, string)
   int line, column;
   char *string;
```

Draw a *string* originating at (*line, column*) in the given *color*. Fonts are fixed pitch. The symbols W_CHAR_WIDTH and W_CHAR_HEIGHT are the width and height of the font in pixels. (Currently, these are 6 and 10). (Macros available - '_b', '_c' and '_xor'.)

```
w_put_characters (window, x, y, color, string)
   int x, y;
   char *string;
```

Draw a *string* originating at (*x,y*) in the given *color*. The origin is the bit position of the upper left corner of the first character to be drawn. (Macros available - '_b', '_c' and '_xor'.)

Positioning and erasing the character cursor.

```
w_position_cursor (window, line, col)
  int line, co;
```

Position the character cursor at the specified (line, col) character position. The cursor is drawn in exclusive or mode.

```
w_erase_cursor (window, line, col)
  int line, col;
```

Erase the character cursor at the specified (line, col) character position.

Scrolling and erasing lines.

```
w_scroll_vertically (window, line, number_of_lines, amount, color)
  int line, number_of_lines;
  int amount;
```

A region of the screen bound by *line* and *number_of_lines* is scrolled vertically the given *amount*, with positive *amounts* being up, and negative down. The fill is given in *color*. Scroll amounts larger than the size of the region are allowed and result in the whole region being blanked. (Macros available - '_b' and '_c'.)

```
w_scroll_horizontally (window, line, col, number_of_cols, amount, color)
  int line, col, number_of_lines;
  int amount;
```

A portion of the given *line* is scrolled horizontally, positive amounts being to the left, negative to the right. _*color* and treatment of large scroll amounts are as above. (Macros available - '_b' and '_c'.)

```
w_erase_lines (window, line, col, number_of_lines, color)
  int line, col;
  int number_of_lines;
```

Erase the line segments starting at (*line, col*) for the *number_of_lines* indicated. The line will be erased only partially, starting at *col*. The fill is given by *color*. (Macros available - '_b' and _c.)

Bit and vector graphics.

```
w_draw_vector (window, x1, y1, x2, y2, color)
  int x1, y1;
  int x2, y2;
```

Draw a vector between the coordinates (*x1, y1*) and (*x2, y2*) in the given color. (Macros available - '_b', '_c' and '_xor'.)

```
w_put_bits (window, x, y, number_of_bits, color, bit_pattern)
  int x, y;
  int number_of_bits;
  unsigned char bit_pattern[];
```

Draw the given *number_of_bits* from the array *bit_pattern* on the screen, starting at pixel point (*x,y*) in the given *color*. Only "lines" of pixels may be specified; regions are not possible in a single call. (Macros available - '_b', '_c' and '_xor'.)

Rectangle graphics.

```
w_clear_rectangle (window, x, y, x_extent, y_extent, color)
   int x,y;
   int x_extent, y_extent;
```

Clear the defined rectangular area, starting at (*x,y*) the pixel amount in *x_extent, y_extent* in the given *color*. (Macros available - '_b', '_c' and '_xor'.)

```
w_scroll_rectangle (window, x, y, x_extent, y_extent, direction, amount, color)
   int x,y;
   int x_extent, y_extent;
   int direction, amount;
```

A rectangular area of pixels is scrolled in the *direction* given for the *amount* given. The number of pixels to scroll (amount) may be negative, which reverses the direction. The amount may be greater than the extent of the region, in which case the entire region is set to the fill *color*. The directions to scroll are given as the constants WM_UP, WM_DOWN, WM_LEFT, WM_RIGHT. (Macros available - '_b' and '_c'.)

```
w_raster_copy (window, x_src, y_src, x_dest, y_dest, x_extent, y_extent)
   int x_src,y_src;
   int x_dest, y_dest;
   int x_extent, y_extent;
```

Copy a rectangular block of bits from one area of the window to another (possibly overlapping) area of the window. The x and y *src* and *dest* points are the lower left corner of the source and destination blocks respectively; the x. y *extents* give the size of the block to be moved. Both the source and destination rectangles must fit completely inside the window.

Miscellaneous output requests.

```
w_ring_bell (window)
```

Ring the bell

A.7 Program control of screens.

The window manager may be directed to create windows or to return information about the workstation screen. Note that these calls are prefixed by *wm_* (which stands for Window Manager) as they are not window-specific. The first parameter in all these calls is *window_manager* which is of the type j_process_id. If you already have an existing window parameter, you may use the call *w_get_manager (window)* in place of the first parameter.

Information about the screen.

```
wm_get_screen_size (window_manager, x_size, y_size)
   int *x_size, *y_size;
```

Returns the absolute size of the workstation screen in pixels *(x_size, y_size)*.

```
int wm_get_window_list (window_manager, list, max_windows)
   int list[];
   int max_windows;
```

Returns a sequence of integers in the array *list*, each one being the window identifier of a window on the workstation screen. The capacity of the array is given in *max_windows*. The actual number of windows on the screen is returned by the function.

Creating a New Window.

```
int wm_create (window_manager, x1, y1, x2, y2)
   int x1, y1, x2, y2;
```

Create a window on the screen with the absolute screen coordinates given in *(x1, y1)* and *(x2, y2)* and return the window identifier of the new window (0 if the call failed). Security must be OFF for this call to work.

```
int wm_init_window_creation (window_manager, default_x_size, default_y_size)
   int default_x_size, default_y_size;
```

Place the user into window creation mode, with the default window size as given, and return the window identifier of the new window (0 if the call failed).

Manipulating the Mouse Cursor.

```
wm_move_cursor (window_manager, x, y)
   int x, y;
```

Move the mouse cursor to the absolute screen coordinates given in *(x, y)*and update the currently 'active' window accordingly. The call will not work unless the security flag is set to OFF.

A.8 Program control of windows.

The following window manager calls are all non-standard as they all manipulate windows, a task normally done by the user. As the Jade window interface stresses user control of windows, you may find it a bit difficult to use some of these calls without distrupting the users view of the system.

All calls will not work unless the security bit in the Console window OPTIONS is turned OFF. This security bit may be toggled by downloaded programs; they can call a routine named 'flip_security', which toggles the security bit and returns the now current state of the flag. See the Jade manual for further information.

Some of the screen routines described in "program control of windows" (section I.a.7) will be handy for finding out necessary screen information.

Changing the window appearance.

w_bury (window)

Bury the identified window to the bottom of the stack, exactly as if done via the 'THIS WINDOW' menu. The call will not work unless the security flag is set to OFF.

w_raise (window)

Raise the identified window to the top of the stack, exactly as if done via the 'THIS WINDOW' menu. The call will not work unless the security flag is set to OFF.

w_reverse (window)

Reverse the background colour of the identified window, exactly as if done via the 'THIS WINDOW' menu. The call will not work unless the security flag is set to OFF.

w_destroy (window)

Destroy the identified window, exactly as if done via the 'THIS WINDOW' menu. The call will not work unless the security flag is set to OFF.

```
w_place (window, x1, y1, x2, y2)
   int x1, y1, x2, y2;
```

Place and resize the current window so that it resides on the screen with its lower-left and upper-right corners at the given *(x1,y1) (x2,y2)* points. The behaviour duplicates exactly that of doing a place via the 'THIS WINDOW' menu. The call will not work unless the security flag is set to OFF.

A.9 Example Programs.

This section contains examples of how to use the facilities described earlier. The programs on the following pages illustrates how one initializes a window, makes requests of the window manager to set the title and put up a menu, handles input events, and how to do output.

Note the general layout of the programs. First, you must include the appropriate include file *#include <tools.h>*. Second, you must find the window manager and window identifier. After that, you can do what you want!

Example 1.

There are two very similar programs: trirec_vax and trirec_cv which do identical tasks but run on the vax or the corvus respectively. Both programs change the title bar and allow the user to select a menu option which draws a triangle or a rectangle.

Note that there are much simpler ways to write the program if one uses the advanced routines in later sections. However, we advise you to study the example to aid your understanding of this section.

The examples are located in the following sources. Executable versions have the same name with the ".c" removed.

```
~ jade/examples/toolbox/Trirec/trirec_vax.c
~ jade/examples/toolbox/Trirec/trirec_cv.c
```

You can compile the sources by copy the above files into your directory. As with all jade software, you must use the cc and ccx defined in /usr/local/jade/bin/ to compile your programs. Then type (for the vax and corvus versions respectively):

```
cc     trirec_vax.c -o trirec_vax          -ltools -lvterm -ljipc/2
ccx    trirec_cv.c -o trirec_cv            -ltools -ljipc/2
```

To run the program on the vax, type *trirec_vax* in a vterm window. On the corvus, create an *Other* window and type in the full pathname of *trirec_cv.*

Example 2.

Another program exists which demonstrates most of the window manipulation routines. To run it, security must be set to OFF (by selecting security from the OPTIONS menu in the Console window). The program is intensionaly disconcerting to make you realize that program control of windows should not be abused. No corvus version is present, although one could easily be written. The source is in:

```
~ jade/examples/toolbox/Manipulate/manipulate_windows.c
```

and can be compiled with:

```
cc manipulate_windows.c -o manipulate_windows -ltools -lvterm -ljipc/2
```

To run the program, just type *manipulate_windows* in a Vterm window and go for the ride!

```
/* Source in ~jade/examples/toolbox/Trirec/trirec_vax.c                       */
/* This program demonstrates menu and help creation, simple event handling,   */
/* and output requests.  Similar to example in IIIB of the Jade Manual.       */
#include <stdio.h>
#include <jipc/2.h>
#include <windows.h>
#include <vterm.h>
#include <tools.h>

WINDOW  window;                 /* Window to be initialize                    */
int     x_size;                 /* Size of window in X direction              */
int     y_size;                 /* Size of window in Y direction              */
char    old_window_title[80];   /* Original title of the window               */
main () {
    initialize ();              /* Set up the window                          */
    while (process_event ()) ;  /* Process events until FALSE returned        */
    clean_up ();                /* Clean up the mess                          */
}

/* Find the window manager and the window id we talk to, set up               */
/* the pop-up menus, get the initial size of the window and clear it.         */
initialize () {
    j_process_id wm;                            /* The window manager         */
    int wi;                                     /* The window id              */

    if (!vterm_tty (2)) {                       /* Run only in vterm window */
        fprintf (stderr, "Not in a vterm window\n");
        exit (1);
    }

    j_enter_system(0, "TRIREC");            /* Jipc initialization stuff      */
    vterm_disable_input (2, &wm, &wi);      /* Dont want vterm to interfere   */
    vterm_direct_output (2, &wm, &wi);      /* Talk to real window manager    */

    window = w_init (wm, wi);               /* Initialize window              */
    w_buffered (window);                    /* Buffer the window output       */

                                            /* Add help message and menu      */
    w_add_simple_help (window,
     "TRIREC|This example program lets you draw triangles & rectangles.");
    w_add_simple_menu (window,
     "TRIREC|Triangle|Rectangle|Quit",
     "- TRIREC|The proper menu choice draws a triangle, rectangle, or quits");
    w_item_help (window, "TRIREC", "Triangle",
     "-- TRIANGLE|Selecting this item will cause a triangle to be drawn");
    w_item_help (window, "TRIREC", "Rectangle",
     "-- RECTANGLE|Selecting this item will cause a rectangle to be drawn");
    w_item_help (window, "TRIREC", "Quit",
     "-- QUIT|Terminates this program, returning you to Vterm.");

    w_get_window_size (window, &x_size, &y_size);     /* Get window size      */
    w_get_title (window, old_window_title);           /* Change the title     */
    w_set_title (window, "TRIREC - an example program");
    w_clear_b (window);                               /* Clear  window        */
}

/* Clean up the mess.  Delete the menus and the help pop-up windows,          */
/* turn vterm back on, and clear the screen.                                  */
clean_up () {
    w_clear_b (window);                               /* Clear  window        */
    w_delete_help (window, "TRIREC");
    w_delete_menu (window, "TRIREC");
    w_set_title (window, old_window_title);
    vterm_enable_input (2);
}

/* Process an input event. SIZE_SET events are noted, DESTROY events          */
/* cause an exit, and MENU_SELECTION events draws a triangle, rectangle,      */
/* or returns 0 for a Quit. Other events are ignored.                         */
int process_event () {
    char    ignore_string[20];                  /* Parameters we don't need */
    char    item[20];                           /* Item on selected menu      */
    int     ignore_int;                         /* Parameters we don't need */

    switch ( w_get_next_event (window) ) {      /* Get the input event        */
        case WM_SIZE_SET:                       /* Size has been changed      */
            w_get_point (window, &x_size, &y_size);
            break;
        case WM_DESTROY:                        /* Window destroyed           */
            exit (0);
        case WM_MENU_SELECTION:                 /* Menu selection             */
            w_get_menu (window, &ignore_int, ignore_string, item, &ignore_int);
            w_clear (window);
            if (strcmp (item, "Triangle") == 0) {
                w_draw_vector_c (window, x_size/2, 10, 10, y_size-10);
                w_draw_vector_c (window, 10, y_size-10, x_size-10, y_size-10);
                w_draw_vector_c (window, x_size-10, y_size-10, x_size/2, 10);
            }
            else if (strcmp (item, "Rectangle") == 0) {
                w_draw_vector_c (window, 10, 10, x_size-10, 10);
                w_draw_vector_c (window, x_size-10, 10, x_size-10, y_size-10);
                w_draw_vector_c (window, x_size-10, y_size-10, 10, y_size-10);
                w_draw_vector_c (window, 10, y_size-10, 10, 10);
            }
            else if (strcmp (item, "Quit") == 0) return (FALSE);
            break;
    }
    return (TRUE);
```

```
/* Source in ~jade/examples/toolbox/Trirec/trirec_cv.c                       */
/* This program demonstrates menu and help creation, simple event handling,  */
/* and output requests.  Similar to example in IIIB of the Jade Manual.      */

#include <jipc/2.h>
#include <windows.h>
#include <tools.h>

WINDOW  window;                 /* Window to be initialize                   */
int     x_size;                 /* Size of window in X direction             */
int     y_size;                 /* Size of window in Y direction             */

main () {
    initialize ();                          /* Set up the window             */
    while (TRUE) process_event ();          /* Process events forever        */
}

/* Find the window manager and the window id we talk to, set up              */
/* the pop-up menus, get the initial size of the window and clear it.        */
initialize () {
    j_process_id wm;                        /* The window manager            */
    int wi;                                 /* The window id                 */

    j_initialize ();                        /* Jipc initialization           */
    wm = j_search_locally ("window_manager");
    wi = MY_WINDOW;

    window = w_init (wm, wi);           /* Initialize window                 */
    w_buffered (window);        /* Buffer the window output          */
                                        /* Add help message and menu         */
    w_add_simple_help (window,
     "TRIREC|This example program lets you draw triangles & rectangles.");
    w_add_simple_menu (window,
     "TRIREC|Triangle|Rectangle",
     "- TRIREC|The proper menu choice draws a triangle, or rectangle");
    w_item_help (window, "TRIREC", "Triangle",
     "-- TRIANGLE|Selecting this item will cause a triangle to be drawn");
    w_item_help (window, "TRIREC", "Rectangle",
     "-- RECTANGLE|Selecting this item will cause a rectangle to be drawn");

    w_get_window_size (window, &x_size, &y_size);      /* Get window size    */
    w_set_title (window, "TRIREC - an example program");
    w_clear_b (window);                                /* Clear  window      */
    w_draw_string_b (window, 1, 1, "Destroy this window to quit");
}

/* Process an input event. SIZE_SET events are noted, DESTROY events         */
/* cause an exit, and MENU_SELECTION events draws a triangle, rectangle,     */
/* or returns 0 for a Quit. Other events are ignored.                        */

int process_event () {
    char    ignore_string[20];              /* Parameters we don't need */
    char    item[20];                       /* Item on selected menu        */
    int     ignore_int;                     /* Parameters we don't need */

    switch ( w_get_next_event (window) ) {  /* Get the input event          */

        case WM_SIZE_SET:                   /* Size has been changed        */
            w_get_point (window, &x_size, &y_size);
            break;

        case WM_DESTROY:                    /* Window destroyed             */
            exit (0);

        case WM_MENU_SELECTION:             /* Menu selection               */
            w_get_menu (window, &ignore_int, ignore_string, item, &ignore_int);
            w_clear_b (window);
            if (strcmp (item, "Triangle") == 0) {
                w_draw_vector_c (window, x_size/2, 10, 10, y_size-10);
                w_draw_vector_c (window, 10, y_size-10, x_size-10, y_size-10);
                w_draw_vector_c (window, x_size-10, y_size-10, x_size/2, 10);
            }
            else if (strcmp (item, "Rectangle") == 0) {
                w_draw_vector_c (window, 10, 10, x_size-10, 10);
                w_draw_vector_c (window, x_size-10, 10, x_size-10, y_size-10);
                w_draw_vector_c (window, x_size-10, y_size-10, 10, y_size-10);
                w_draw_vector_c (window, 10, y_size-10, 10, 10);
            }
            break;
    }

}
```

```
/* Source in ~jade/examples/toolbox/Manipulate/manipulate_windows.c
 *
 * Runs on the vax only, but simple changes make it downloadable.
 * Illustrates most window and screen manipulation routines.
 * It demonstrates some bad features of window manipulation in
 * the hope that you will use these routines VERY conservatively
 */
#include <stdio.h>
#include <vterm.h>
#include <jipc/2.h>
#include <windows.h>
#include <tools.h>

main () {
    j_process_id wm;                          /* The window manager    */
    int wi;                                   /* The window id         */
    WINDOW window, window1;                   /* Windows we create     */
    int new_window;                           /* Id of created window */
    int x1, y1, x2, y2, color, centre;        /* Info about new window*/
    int sx, sy;                               /* Screen size           */
    int window_list[20], number_of_windows;   /* Windows on the screen*/
    char str[100];                            /* Working variables     */
    int i;

    if (!vterm_tty (2)) {                     /* Must be vterm window */
        fprintf (stderr, "Not a vterm window\n");
        exit (1);
    }
    j_enter_system(2, "Example");             /* Jipc stuff            */
    vterm_disable_input (2, &wm, &wi);
    vterm_direct_output(2, &wm, &wi);

    wm_get_screen_size (wm, &sx, &sy);        /* Get info about screen*/

    new_window = wm_create (wm,0,0, sx/2, sy/2); /* Create a new window */
    if (new_window != 0)                      /*  and tell the user    */
        window = w_init ( wm, new_window);
    else {
        vterm_enable_input (2);
        fprintf (stderr, "You must turn security off in the Console ");
        fprintf (stderr, "window\nto run this program.\n");
        return (-1);
    }
    w_unbuffered(window);
    centre = sy/4;
    w_put_characters_c (window, 10, centre, "I'm a new window!");
    sleep(4);
                                              /* Move the cursor       */
    w_put_characters_c (window, 10, centre, "Watch my cursor move!");
    for (x1 = 10; x1<= 136; x1++)
        wm_move_cursor (wm, x1, centre-7);
                                              /* Reverse the window   */
    w_put_characters_c (window, 10, centre, "I've reverse myself. Fancy! ");
    w_reverse (window);
    sleep(3);
                                              /* Bury/raise the window*/
    w_put_characters_c (window, 10, centre, "Now to bury myself...        ");
    sleep(3);
    w_bury (window);
    sleep(2);
    w_raise (window);
                                              /* Move the window       */
    w_put_characters_c (window, 10, centre, "Time to move!                 ");
    sleep(3);
    w_get_info (window, &x1, &y1, &x2, &y2, &color);
    w_place (window, sx/2, sy/2, sx, sy);
    w_put_characters_c (window, 10, centre, "Here I am!                    ");
    sleep(3);
                                              /* Reverse all windows  */
    w_put_characters_c (window, 10, centre, "Lets reverse all windows     ");
    sleep(3);
    number_of_windows = wm_get_window_list (wm, window_list, 20);
    for (i = 0; i < number_of_windows; i++){
        window1 = w_init (wm, window_list[i]);
        w_reverse (window1);
        w_end (window1);
    }
                                              /* Put into place mode  */
    w_put_characters_c (window, 10, centre, "Ok, create a new window:     ");
    w_ring_bell (window);
    new_window = wm_init_window_creation (wm, 150, 100);
    if (new_window != 0) {
        w_unbuffered (window1);
        window1 = w_init (wm, new_window);    /* and make a new window*/
        w_draw_string_c (window1, 2, 2, "Hi there, sugar buns!");
        sleep (5);
        w_destroy (window1);
        w_end (window1);
    }
                                              /* So long, and destroy */
    w_put_characters_c (window, 10, centre, "Thats all. Bye!              ");
    sleep(5);
    w_destroy (window);
    w_end (window);
    vterm_enable_input (2);
}
```

B.

Manipulating Cursors, Windows And Menus

Revision of 30 June 1985

B. MANIPULATING CURSORS, WINDOWS & MENUS

The first section provided the primitives for manipulating many window manager facilities. Unfortunately, these primitives are very low level and require much work on the part of the programmer for setting up conceptually simple entities.

This section provides a higher level interface to some of these tasks. Specifically,

- the window cursor may be changed to a pre-defined cursor shape in a single call
- window titles and help menus may be defined in a structure and installed in a single call
- pop-up menus and their associated help windows may be defined in a structure and installed in a single call

But remember, there is no such thing as a free lunch. There is a tradeoff of ease of use versus power. If you wish to do something outside the constraints of these high level routines, you must go back to the first tool box.

B.1. Changing The Cursor

The routines offered here allow changing or saving of the cursor in a specific window. There are many applications for changing the cursor. For example, a time-consuming task may change the cursor to an hourglass and back again afterwards. Different applications may ask for a cursor related to it, such as the pencil one for sketching.

Window has its usual meaning. *Cursor_type* is one of the sets of cursors defined as constants. (The complete list is provided below). Other cursors not in this list may be included at your request.

The Routine.

```
w_change_cursor (window, cursor_type)
   WINDOW window;
   int cursor_type;
```

Change the cursor to the one defined by cursor_type (see below)

Cursor types.

The following cursor constants are currently available.

DEFAULT_CS	*Standard left-leaning arrow*
LEFT_ARROW_CS	*Standard left-leaning arrow (as above)*
TARGET_CS	*Target*

GLASSES_CS	*Glasses*
PENCIL_CS	*Writing Pencil*
HOURGLASS_CS	*Hourglass*
GROUCHO_CS	*Groucho Marx*
CORNER_LL_CS	*Lower left corner*
CORNER_LR_CS	*Lower right corner*
CORNER_UL_CS	*Upper left corner*
CORNER_UR_CS	*Upper right corner*
CROSS_CS	*Cross*
X_CS	*X*
CONFIRM_CS	*Mouse with middle button highlit*
STASH_CS	*The system stash cursor*

Example use.

The following bit of code will change the cursor into an hourglass (telling the user to wait), execute some arbitrary code, and then restore the default cursor.

```
w_change_cursor (window, HOURGLASS_CS)          Cursor becomes a pencil
      -- Your code here --                      Execute a long operation
w_change_cursor (window, DEFAULT_CS)            Restore the default cursor
```

B.2. Setting Up A Window

When application programs are run, they will normally change the title of the window and add a help message describing the window's function. It is extremely tedious to lay out the window by using the first toolbox. This series of routines and structures overcome this problem.

The approach given here is to define windows by filling in structures. The structure *window_layout* defines the appearance of the window title and the help screen. Once defined, the window can be "layed out" by a simple procedure call.

Once the window is set up, the programmer can change the appearance of the window by altering the structure (for a complete change) or by calling the more primitive routines in toolbox1 (for small changes).

The structure.

The programmer can set up a window by filling in the following structures (this is declared in the include file <tools.h>:

```
typedef struct window_layout {              Window layout information:
      char title_bar [];                    -Title bar of window
      char window_name [];                  -Name of help window
      char window_help [];                  -Contents of help window
} WINDOW_LAYOUT;
```

The Routine.

Once the structure has been filled, the user has the following layout procedure at his disposal.

```
w_layout_window (window, window_appearance)
   WINDOW window;
   WINDOW_LAYOUT *window_appearance;
```

Window_appearance is the address of the structure. The window *title_bar* as identified in the structure is installed. Similarly, a help pop-up window identified by the *window_name* and containing the window *window_help* is installed.

Example use.

We may set up the window described in the examples at the end of Section A.7 by the following code (the complete example is offered at the end of this section):

```
/* fill in the WINDOW_LAYOUT structure described above */
WINDOW_LAYOUT window_appearance = {
   "TRIANGLE/MESSAGE - an example program",
   "TRIANGLE/MESSAGE",
   "Triangles are drawn in one window, a message is drawn in another"
};

main() {
            -- your code here --
     w_layout_window (window, &window_appearance);
            -- your code here --
```

B.3. Setting Up A Menu

When application programs are run, they will normally create one or more pop-up menus with all their associated help messages. It is extremely tedious to create these menus by using the first toolbox. This series of routines and structures overcome this problem.

The approach given here is to define menus by filling in structures (as in the previous section). The structure *menu_layout* defines the name of the menu and the contents of the help screen. The name of the help screen is taken to be the name of the menu.

The structure *item_layout* defines an array of items to be associated with a given menu. Each item has as an attribute an item name, the contents of the help screen, a flag for new window creation and default sizes of that window (see w_make_window in A.5.) Once defined, the pop-up menu and all the associated windows can be "layed out" by a simple procedure call. If the programmer wishes to change the menu, he can delete the old menu (w_delete_menu, section A.4), then change the structure and lay it out again.

The structure.

The programmer can set up a menu by filling in the following structures (this is declared in the include file <tools.h>:

```
typedef struct menu_layout {            Menu layout information
      char menu_name [];                -Name of menu & help window
      char menu_help [];                -Contents of help window
} MENU_LAYOUT;

typedef struct item_layout {            Item layout information
      char item_name [];                -Name of item & help window
      char item_help [];                -Contents of help window
      int    new_window_flag;           -Make a new window flag
      int    x_default_size;            -Default (x,y) window size
      int    y_default_size;             in pixels
} ITEM_LAYOUT;
```

The Routine.

Once the structure has been filled, the user has the following layout procedures at his disposal. The first (*w_layout_menu_in_pane*) installs the menu in the indicated panes. The second (*w_layout_menu*) is a macro, which installs the menu over the whole window.

```
w_layout_menu_in_pane (window, menu_appearance, item_appearance, number_items,
                        division_vector, division_mask)
   WINDOW window;
   MENU_LAYOUT *menu_appearance;
   ITEM_LAYOUT item_appearance[];
   int number_items;
   int division_vector;
   int division_mask;
```

Menu_appearance and *item_appearance* are the addresses of the structure and *number_items* is the number of menu items. A pop-up menu labeled with the menu name and a corresponding help window are installed in the panes indicated by *division_vector* and *division_mask*. See section A.5 of this manual or section III.B of the Jade Manual for a description of panes. The menu is filled with items from the items structure, with each item having a corresponding help window. A new window will be made of a given size on an item's selection if that item has the new_window_flag set in the structure.

```
w_layout_menu (window, menu_appearance, item_appearance, number_items)
```

This macro is identical to the one above except that the menu is installed over the whole window.

Useful constants.

The following constants may be used within the *item_layout* structure for defining new window creation and its size.

W_MAKE_WINDOW	*Will make a window in new_window_flag*
W_DEFAULT_COLUMNS	*80 columns in the x_default_size*
W_DEFAULT_LINES	*24 lines in the y_default_size*
W_IGNORE	*Don't make a window*

The *new_window_flag* may be initialized by the definitions W_MAKE_WINDOW or W_IGNORE. The *default_size* of the window may be initialized to W_DEFAULT_COLUMNS, W_DEFAULT_LINES, W_IGNORE or your desired size.

Example use.

We may set up the pop-up menu described in the examples at the end of Section A.7 by the following code (the complete example is offered at the end of this section):

```
    #define NUMB_ITEMS        3

   /* fill in the MENU_LAYOUT structure described above */
    MENU_LAYOUT menu_appearance = {
        "TRI/MESSAGE",
        "The proper menu choice draws a triangle, prints a message or quits."
    };

   /* fill in the ITEM_LAYOUT structure described above */
    ITEM_LAYOUT item_appearance[NUMB_ITEMS] = {
         "Triangle",
         "Selecting this item will cause a triangle to be drawn",
         W_IGNORE, W_IGNORE, W_IGNORE,

         "Message",
         "Asks you to create a new window and writes a message in it",
         W_MAKE_WINDOW, 100, 50,

         "Quit",
         "Terminates this program, returning you to Vterm",
         W_IGNORE, W_IGNORE, W_IGNORE
    };

    main() {
                  -- your code here --
         w_layout_menu (window, &menu_appearance, item_appearance, NUMB_ITEMS);
                  -- your code here --
```

B.4 Example Programs.

This section contains two examples of how to use the facilities described earlier.

Example 1.

The first program used is a variation of the "trirec" program described in the first section and is called *tri_message.c*.

Tri_message.c changes the cursor to an hourglass until the window is initialized. Part of the initialization process is to lay out the window and pop-up menu as described in this section. The first pop-up menu selection draws a triangle in the current window. The second selection asks the user to create a new window and then puts a message in it.

To avoid cluttering the program, the structures are defined in an include file called *tri_message.h*. The example is located in the following sources:

    ~jade/examples/toolbox/Trimessage/tri_message.c

```
˜ jade/examples/toolbox/Trimessage/tri_message.h
```

You can compile the sources in your own directory by:

```
cc    tri_message.c -o tri_message -ltools -lvterm -ljipc/2
```

To run the program on the vax, type *tri_message* in a vterm window. You may wish to write the equivalent down-loadable version as an exercise - it is exactly the same initialization process as in trirec_cv.c (see section A).

Example 2.

The second program introduces a driver which manipulates many example programs, running as independant processes.. It allows the user to create windows and run the examples in them. The interface is quite simple: The user makes a selection from a pop-up menu which asks the user to create a window. Once created, the driver attaches a specific process to the new window. These processes then run independantly.

The source for the driver is in *examples.c*. It uses *ifdefs* so that it can be compiled for either the vax or the corvus (as described below). The executable versions are called *examples_vax* and *examples_cv* respectively. As this driver is secondary to our examples, we do not include a listing in this manual. However, you are invited to browse the the on-line source at your leisure.

Only one sub-process is described here: *cursor.c*. This program puts up a pop-up menu with all the possible cursor types. Selection of a given menu item will change the cursor to the one described. The executable form of this process is in *cursor_vax* and *cursor_cv* for the vax and the corvus respectively. Although these versions must be initialized by the driving program *examples.c*, it is a simple exercise to make them run independantly. The other sub-processes will be described in following sections.

The sources are located in:

```
˜ jade/examples/toolbox/Example_driver/examples.c
˜ jade/examples/toolbox/Example_driver/examples.h
˜ jade/examples/toolbox/Cursor/cursor.c
```

You can compile examples.c for the vax or the corvus by:

```
cc     examples.c -o examples -DVAX -ltools -lvterm -ljipc/2
ccx     examples.c -o examples -ltools
```

The cursor program is compiled in exactly the same name, except that the -D flag is not necessary.

IMPORTANT: If you copy the examples.c program, you must change the pathnames of the processes in the examples.h file to match the ones you want to fire up! If you do not do this, or if you do it incorrectly it will either run the default examples or hang altogether.

```
/* Source in ~jade/examples/toolbox/Trimessage/trimessage.c                   *
 * This program illustrates changing the cursor and how to layout the         *
 * window and pop up menus with high-level functions.  In addition,           *
 * it illustrates how to make a new window and output messages to it          *
 */

#include "tri_message.h"

main () {
    initialize ();                /* Set up the window                        */
    while (process_event ()) ;    /* Process events until FALSE returned      */
    clean_up ();                  /* Clean up the mess                        */
}

initialize () {
    j_process_id wm;                            /* The window manager         */
    int wi;                                     /* The window id              */

    if (!vterm_tty (2)) {                       /* Run only in vterm window */
        fprintf (stderr, "Not in a vterm window\n");
        exit (1);
    }

    j_enter_system(0, "TRIREC");        /* Jipc initialization stuff          */
    vterm_disable_input (2, &wm, &wi);  /* Dont want vterm to interfere       */
    vterm_direct_output (2, &wm, &wi);  /* Talk to real window manager        */

    window = w_init (wm, wi);           /* Initialize window                  */
    w_change_cursor (window, HOURGLASS_CS);    /* Cursor becomes an hourglass */

    w_buffered (window);
    w_get_window_size (window, &x_size, &y_size);
    w_clear_rectangle_b (window, 0, 0, x_size, y_size);
    w_get_title (window, old_window_title);

    /* Layout the window and install the pop-up menus                         */
    w_layout_window (window, &window_appearance);
    w_layout_menu (window, &menu_appearance, item_appearance, NUMB_ITEMS);

    w_change_cursor (window, DEFAULT_CS);  /* Change cursor to the default one */
}

clean_up () {
    w_clear_rectangle_b (window, 0, 0, x_size, y_size);
    w_delete_help (window, window_appearance.window_name);
    w_delete_menu (window, menu_appearance.menu_name);
    w_set_title (window, old_window_title);
    vterm_enable_input (2);
}

int process_event () {
    char    ignore_string[20];                  /* Parameters we don't need */
    char    item[20];                           /* Item on selected menu    */
    int     ignore_int;                         /* Parameters we don't need */
    int     new_window;
    WINDOW  window2;

    switch ( w_get_next_event (window) ) {      /* Get the input event      */

        case WM_SIZE_SET:                       /* Size has been changed    */
            w_get_point (window, &x_size, &y_size);
            break;

        case WM_DESTROY:                        /* Window destroyed         */
            exit (0);

        case WM_MENU_SELECTION:                 /* Menu selection           */
            w_get_menu (window, &ignore_int, ignore_string, item, &new_window);

            if (strcmp (item, item_appearance[0].item_name) == 0) {
                w_clear_rectangle_b (window, 0, 0, x_size, y_size);
                w_draw_vector_c (window, x_size/2, 10, 10, y_size-10);
                w_draw_vector_c (window, 10, y_size-10, x_size-10, y_size-10);
                w_draw_vector_c (window, x_size-10, y_size-10, x_size/2, 10);

            }

            /* Put out a message in the new window                            */
            else if (strcmp (item, item_appearance[1].item_name) == 0) {
                window2 = w_init ( w_get_manager (window), new_window);
                w_draw_string_c (window2, 2, 2, "Hi there, sugar buns!");
                w_end (window2);
            }

            else if (strcmp (item, item_appearance[2].item_name) == 0)
                 return (FALSE);
            break;
    }
    return (TRUE);
}
```

```
/* Source in ~jade/examples/toolbox/Trimessage/trimessage.h                    */

#include <stdio.h>
#include <jipc/2.h>
#include <windows.h>
#include <vterm.h>
#include <tools.h>

WINDOW  window;                 /* Window to be initialize                    */
int     x_size;                 /* Size of window in X direction              */
int     y_size;                 /* Size of window in Y direction              */
char    old_window_title[80];   /* Original title of the window               */

/* What the window should look like */
WINDOW_LAYOUT window_appearance = {
    "TRIANGLE/MESSAGE - an example program",
    "TRIANGLE/MESSAGE",
    "Triangles are drawn in one window, a message is drawn in another"
};

/* The title of the menu and the help message */
MENU_LAYOUT menu_appearance = {
    "TRI/MESSAGE",
    "The proper menu choice draws a triangle, prints a message or quits."
};


#define NUMB_ITEMS    3                 /* The number of menu items           */

/* The name and help message of each menu item plus the new window state      */
ITEM_LAYOUT item_appearance[NUMB_ITEMS] = {
    "Triangle",
    "Draw a triangle in this window",
    W_IGNORE, W_IGNORE, W_IGNORE,

    "Message",
    "Asks you to create a new window and writes a message in it",
    W_MAKE_WINDOW, 200, 50,

    "Quit",
    "Terminates this program, returning you to Vterm",
    W_IGNORE, W_IGNORE, W_IGNORE
};
```

```
/* ==========================================================
 * Source in ~jade/examples/toolbox/Cursor/cursor.c
 *
 * This is a CHILD process called from another program!
 * This  example program illustrates the different cursor types.
 * ==========================================================
 */
#include "cursor.h"

main () {
    initialize ();
    while (TRUE) process_event ();
}

initialize () {
    j_process_id temp_wm;
    int temp_wi;

    j_initialize();                              /* Jipc stuff to create */
    j_send (j_parent_process() );                /* the child in a window*/
    j_reset();
    temp_wm = j_getp();
    temp_wi = j_geti();

    window = w_init (temp_wm, temp_wi);          /* Set up the window    */
    w_layout_window(window, &w_layout);
    w_layout_menu(window, &menu2, items2, NUM_MENU2_ITEMS);
    w_layout_menu(window, &menu1, items1, NUM_MENU1_ITEMS);
}

/* Events are processed until the window is destroyed.
 * Only menu selections are looked at.
 */
process_event () {
    char    menu_chosen[20];    /* A parameter we don't need            */
    int     ignore_int;         /* A parameter we don't need            */
    char    item_chosen[20];    /* The item on the menu selected        */

    switch (w_get_next_event (window)) {        /* Get the event        */

        case WM_MENU_SELECTION:
            w_get_menu (window, &ignore_int, menu_chosen, item_chosen, &ignore_int);
            if (strcmp (menu_chosen, menu1.menu_name) == 0) {
                if (strcmp (item_chosen, items1[0].item_name) == 0)
                    w_change_cursor (window, DEFAULT_CS);
                else if (strcmp (item_chosen, items1[1].item_name) == 0)
                    w_change_cursor (window, TARGET_CS);
                else if (strcmp (item_chosen, items1[2].item_name) == 0)
                    w_change_cursor (window, GLASSES_CS);
                else if (strcmp (item_chosen, items1[3].item_name) == 0)
                    w_change_cursor (window, PENCIL_CS);
                else if (strcmp (item_chosen, items1[4].item_name) == 0)
                    w_change_cursor (window, HOURGLASS_CS);
                else if (strcmp (item_chosen, items1[5].item_name) == 0)
                    w_change_cursor (window, GROUCHO_CS);
                else if (strcmp (item_chosen, items1[6].item_name) == 0)
                    w_change_cursor (window, CORNER_LL_CS);
                else if (strcmp (item_chosen, items1[7].item_name) == 0)
                    w_change_cursor (window, CORNER_LR_CS);
                else if (strcmp (item_chosen, items1[8].item_name) == 0)
                    w_change_cursor (window, CORNER_UL_CS);
                else if (strcmp (item_chosen, items1[9].item_name) == 0)
                    w_change_cursor (window, CORNER_UR_CS);
                else if (strcmp (item_chosen, items1[10].item_name) == 0)
                    w_change_cursor (window, CROSS_CS);
            }
            else if (strcmp (menu_chosen, menu2.menu_name) == 0) {
                if (strcmp (item_chosen, items2[0].item_name) == 0)
                    w_change_cursor (window, X_CS);
                else if (strcmp (item_chosen, items2[1].item_name) == 0)
                    w_change_cursor (window, CONFIRM_CS);
                else if (strcmp (item_chosen, items2[2].item_name) == 0)
                    w_change_cursor (window, STASH_CS);
            }
            break;

        case WM_DESTROY:                    /* Window destroyed. Get out    */
            exit (0);
    }
}
```

```
/* ===========================================================
 * Source in ~jade/examples/toolbox/Cursor/cursor.h
 * =========================================================== */
#include <jipc/2.h>
#include <windows.h>
#include <tools.h>

#define NUM_MENU1_ITEMS         11
#define NUM_MENU2_ITEMS         3

WINDOW_LAYOUT w_layout = {      /* Window title and help screen          */
 "Cursor",
 "Cursor",
 "This demonstrates the various cursors available in this package.\
  Select the cursor from a pop-up menu.  Destroy the window to quit."
};

MENU_LAYOUT menu1 = {
 "Cursors [1]",
 "This is the first menu that allows you to change the current\
 window cursor."
 };

MENU_LAYOUT menu2 = {
 "Cursors [2]",
 "This is the second menu that allows you to change the current\
 window cursor."

 };

ITEM_LAYOUT items1[NUM_MENU1_ITEMS] = {
 "Default",
 "DEFAULT_CS: The default cursor (left leaning arrow)",
 W_IGNORE, W_IGNORE, W_IGNORE,
 "Target",
 "TARGET_CS: A target cursor",
 W_IGNORE, W_IGNORE, W_IGNORE,
 "Glasses",
 "GLASSES_CS: A pair of spectacles (can indicate reading from disc)",
 W_IGNORE, W_IGNORE, W_IGNORE,
 "Pencil",
 "PENCIL_CS: A drawing pencil (can indicate writing to disc or sketch mode)",
 W_IGNORE, W_IGNORE, W_IGNORE,
 "Hourglass",
 "HOURGLASS_CS: An hourglass (can indicate lengthy operation to user)",
 W_IGNORE, W_IGNORE, W_IGNORE,
 "Groucho",
 "GROUCHO_CS: A picture of Groucho Marx (you decide what to do with this!)",
 W_IGNORE, W_IGNORE, W_IGNORE,
 "Lower left",
 "CORNER_LL_CS: A Lower left corner",
 W_IGNORE, W_IGNORE, W_IGNORE,
 "Lower right",
 "CORNER_LR_CS: A Lower right corner",
 W_IGNORE, W_IGNORE, W_IGNORE,
 "Upper left",
 "CORNER_UL_CS: An upper left corner",
 W_IGNORE, W_IGNORE, W_IGNORE,
 "Upper right",
 "CORNER_UR_CS: An upper right corner",
 W_IGNORE, W_IGNORE, W_IGNORE,
 "Cross",
 "CROSS_CS: A cross",
 W_IGNORE, W_IGNORE, W_IGNORE
};

ITEM_LAYOUT items2 [NUM_MENU2_ITEMS] = {
 "X",
 "X_CS: an X shape",
  W_IGNORE, W_IGNORE, W_IGNORE,
 "Confirm",
 "CONFIRM_CS: A mouse with the middle button highlit.",
  W_IGNORE, W_IGNORE, W_IGNORE,
 "Stash",
 "STASH_CS: The system stash cursor",
  W_IGNORE, W_IGNORE, W_IGNORE

};

WINDOW  window;
```

C.

The Rectangle

Revision of 20 Jan. 1985

C. THE RECTANGLE

The rectangle may be used in many ways. At its simplest, it is merely a rectangle which can be positioned by the programmer or the user, outlined, erased, filled, emptied and queried. Additionally, it can be selected - a point press in the rectangle can be detected by the applications program and acted upon. You will find the rectangle useful for many situations. It is also used as a primitive for many higher-level tools in the toolbox..

C.1 Using The Rectangle.

All rectangle routines begin with the letter 'r_' (for rectangle). All variables are integers, with the exception of window which is of the type WINDOW, and rectangle which is of the type RECTANGLE.

Initialization

Before use, the rectangle must be initialized (*r_init*). As part of the initialization procedure, the currently active window must be supplied. A pointer identifying the rectangle instance is returned from this call, and is declared as a RECTANGLE. All subsequent calls must use it.

Alternatively, you may initialize a new rectangle by copying an old one (*r_copy*). This is useful if you wish to have many similar rectangles differing by only a parameter or two.

The rectangle must then be positioned within the window by supplying the pixel coordinates of the diagonal defining the rectangle (*r_position*). It is the programmer's responsibility to ensure that this area will not be overwritten, and to reposition the rectangle when the window changes size.

Borders may be set at any time (*r_set_borders* or *r_set_all_borders*). These borders are drawn inside the given positions. Border sizes larger than the rectangle are truncated to the size of the rectangle.

Finally, a fill margin may be designated (*r_set_fill_margin*). This has the effect of leaving a margin between the fill area and the border.

RECTANGLE r_init (window)

> Initialize and return a pointer to a rectangle. The defaults position the rectangle between on the zero diagonal (0,0) (0,0) in the window without drawing it, sets the borders to 1 pixel all around and the fill margin to 0, and sets the fill state to FALSE.

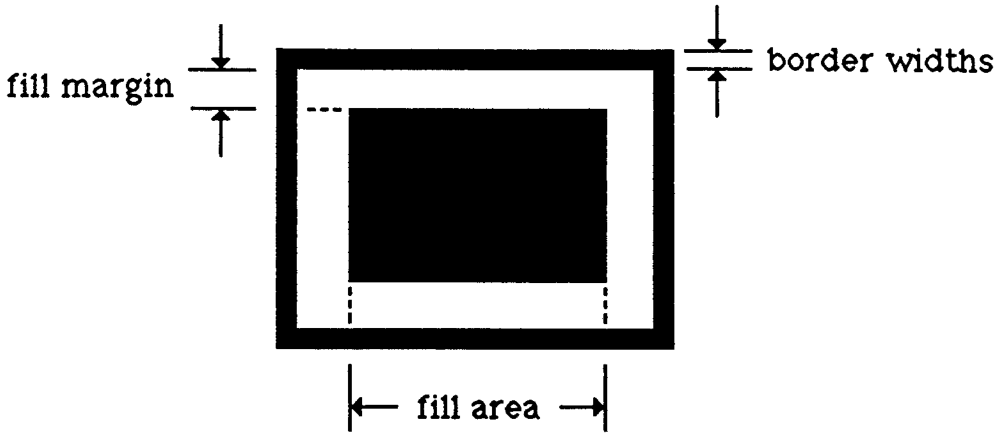
RECTANGLE r_copy (old_rectangle)

> Return a pointer to a new rectangle which is an exact copy of the old rectangle.

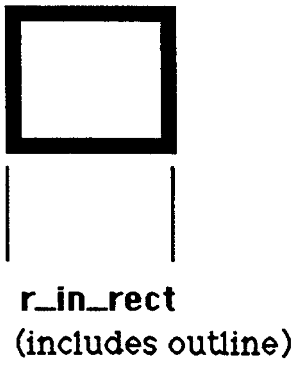
r_position (rectangle, x1, y1, x2, y2)

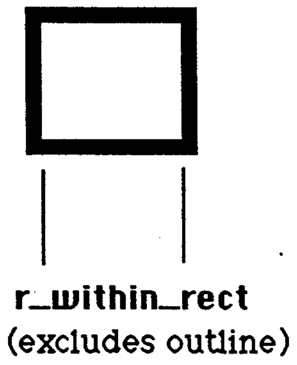
The Rectangle

Architecture:



Using the rectangle:





C. The Rectangle

Position the rectangle between the raster points *(x1,y1)* and *(x2,y2)*

r_set_borders (rectangle, left_border, right_border, top_border, bottom_border)

Set the borders of the rectangle to be the pixel widths given in ***left_border***, ***right_border***, ***top_border*** and ***bottom_border***.

r_set_all_borders (rectangle, border)

Set all borders of the rectangle to be the pixel width of ***border***.

r_set_fill_margin (rectangle, fill_margin)

Set the space between the border and the fill area to the pixel distance given in ***fill_margin***.

Drawing The Rectangle

Nothing will be visible until the rectangle is drawn. The rectangle may be outlined (***r_outline***) or filled. Fills may be exclusive of the outline (***r_fill***), or inclusive -- ie it is not necessary to outline the rectangle if you are going to fill all of it ***r_fill_all***). Once filled, the rectangle may be erased by either leaving the outline intact (***r_erase***) or by erasing the rectangle completely (***r_erase_all***) . Additionally, the rectangle may be 'filled' in exclusive-or mode (***r_fill_xor*** or ***r_fill_all_xor***), which is useful if you wish to toggle the color of the rectangle contents.

Many applications will probably requre knowledge of the current filled or empty state. We may query the rectangle as to its current filled state (***r_is_filled***). If the rectangle is filled a TRUE is returned. Note that drawing in exclusive-or mode toggles this value.

r_outline (rectangle)
Outline the rectangle perimiter

r_fill (rectangle)
Fill the rectangle in, excluding the borders.

r_fill_all (rectangle)
Fill the rectangle in, including the borders.

r_fill_xor (rectangle)
Fill the rectangle in exclusive-or mode, excluding the borders

r_fill_xor_all (rectangle)
Fill the rectangle in exclusive-or mode, including the borders.

r_erase (rectangle)
Clear the inside of the rectangle, leaving the borders intact.

r_erase_all (rectangle)
Clear the rectangle, excluding the borders.

int r_is_filled (rectangle)
Returns TRUE if the rectangle is filled, else FALSE

Actively Using The Rectangle

When a point event is detected, the programmer can detect if the point occurred in the rectangle. A TRUE is returned if it is, otherwise FALSE. Depending on which routine is called, the rectangle area can be considered as the complete rectangle including the border (*r_in_rectangle*) or just the inside of the rectangle (*r_within_rectangle*).

int r_in_rectangle (rectangle, x, y)

> Indicate if the *(x,y)* point is in the rectangle by returning TRUE or FALSE. This includes the borders.

int r_within_rectangle (rectangle, x, y)

> Indicate if the *(x,y)* point is in the rectangle by returning TRUE or FALSE. This excludes the borders.

Querying the Rectangle

All rectangle characteristics may be obtained by the applications programmer at any time. These include the window the rectangle is linked to (*r_get_window*), and the coordinates of the corners (*r_get_corners*) or each of the sides (*r_get_top, r_get_bottom, r_get_left, r_get_right*). All border widths are obtainable in one lump (*r_get_borders*) or individually (*r_get_left_border, r_get_right_border, r_get_top_border, r_get_bottom_border*). The fill margin is also retrievable (*r_get_fill_margin*).

Additionally, mathematical rectangle functions are available to find its height (*r_get_height*), width (*r_get_width*) and area (*r_get_area*).

WINDOW r_get_window (rectangle)
    return the window the rectangle is linked to.

r_get_corners (rectangle, left, bottom, right, top)
    return the (*left, bottom*) and (*right, top*) points of the rectangle.

int r_get_left (rectangle)
int r_get_right (rectangle)
int r_get_top (rectangle)
int r_get_bottom (rectangle)
    return the pixel coordinate of the described rectangle boundary.

r_get_borders (rectangle, left, bottom, right, top)
    return the *left, bottom, right* and *top* border widths.

int r_get_left_border (rectangle)
int r_get_right_border (rectangle)
int r_get_top_border (rectangle)
int r_get_bottom_border (rectangle)
    return the pixel widths of the described border.

int r_get_fill_margin (rectangle)
    return the pixel width of the fill margin.

int r_get_height (rectangle)
int r_get_width (rectangle)

return the height or width of the rectangle in pixels.

int r_get_area (rectangle)
return the area of the rectangle in pixels.

Ending it All

When the rectangle is no longer needed, the programmer should close it (*r_end*) to release the space.

r_end (rectangle)
End the rectangle

C.2 Example Program

This section contains an example of how to use the facilities described earlier. The program ***rectangle.c*** is a process spawned by a driving program ***examples.c*** (see section B.4). Similar to cursor.c, it cannot be run independantly although the program modification to do so is simple.

This simple program will put up a rectangle in the center of a window. A point press in the rectangle region will change the color and print out on/off messages.

The executable process is called ***rectangle_vax*** and ***rectangle_cv*** for the vax and the corvus respectively. Compilation is similar to that described in section B.4. The sources and executable processes can be found in:

```
~ jade/examples/toolbox/Rectangle/rectangle.c
~ jade/examples/toolbox/Rectangle/rectangle_vax
~ jade/examples/toolbox/Rectangle/rectangle_cv
```

```
/* ============================================================
 * Source in ~jade/examples/toolbox/Rectangle/rectangle.c
 *
 * This is a CHILD process called from another program!
 * This  example program puts up a rectangle on the screen.
 * The rectangle is "on" when it is filled and "off" when it isn't.
 * An on/off message is shown.
 * ============================================================
 */
#include <jipc/2.h>
#include <windows.h>
#include <tools.h>

#define X1      (x_size/2 - 15)                /* Rectangle is 30 x 30 */
#define X2      (x_size/2 + 15)                /* in middle of window  */
#define Y1      (y_size/2 - 15)
#define Y2      (y_size/2 + 15)

WINDOW_LAYOUT w_layout = {      /* Window title and help screen         */
 "Rectangle",
 "Rectangle",
 "This demonstrates a toggle rectangle.  When the rectangle is pressed,\
  its color is flipped and an on/off message is printed.\
  Destroy the window to quit."
};
                                /* GLOBALS (Yuk)                        */
WINDOW  window;                 /* Window manager and window id         */
RECTANGLE rectangle;            /* Toggle rectangle                     */
int x_size, y_size;             /* Current size of the window           */

main () {
    initialize ();
    while (TRUE) process_event ();
}

initialize () {
    j_process_id temp_wm;
    int temp_wi;

    j_initialize();                            /* Jipc stuff to create */
    j_send (j_parent_process() );              /* the child in a window*/
    j_reset();                                 /* Don't alter it unless*/
    temp_wm = j_getp();                        /* you know what you're */
    temp_wi = j_geti();                        /* doing.               */

    window = w_init (temp_wm, temp_wi);        /* Set up the window    */
    w_layout_window(window, &w_layout);
    w_buffered (window);

    rectangle = r_init (window);               /* Initialize rectangle */
    r_set_all_borders  (rectangle, 2);         /* A border of 2 pixels */
    r_set_fill_margin  (rectangle, 1);         /* Margin on the inside */
}

/* Events are processed until the window is destroyed.
 * A size change redraws the rectangle or prints an error
 * message if the window is too small
 * A point press toggles the rectangle if the point is in it.
 */
process_event () {
    int x, y;
    char message[10];

    switch (w_get_next_event (window)) {              /* Get the event */

        case WM_SIZE_SET:
            w_get_point (window, &x_size, &y_size);
            w_clear_rectangle_b (window, 0, 0, x_size, y_size);

            r_position (rectangle, X1, Y1, X2, Y2);
            if (r_is_filled (rectangle) ) {
                r_fill_all (rectangle);
                w_put_characters_b (window, X1, Y1-10, "On ");
            }
            else {
                r_outline (rectangle);
                w_put_characters_b (window, X1, Y1-10, "Off");
            }
            break;

        case WM_POINT_DEPRESSED:
            w_get_point (window, &x, &y);

            if(r_in_rect (rectangle, x, y)) {         /* Toggle it    */
                if (r_is_filled (rectangle))
                    strcpy (message, "Off");
                else
                    strcpy (message, "On ");
                r_fill_xor (rectangle);
                w_put_characters_b (window, X1, Y1-10, message);
            }
            break;

        case WM_DESTROY:                  /* Window destroyed. Get out    */
            r_end (rectangle);
            exit (0);
    }
}
```

D.

The Slide Potentiometer

Revision of 30 June 1985

D. THE SLIDE POTENTIOMETER

The slide potentiometer is a valuator, resembling a slide bar volume control on a stereo. This control offers the programmer an interface tool for accepting a user-selected value from a given range. It operates by moving a positioning bar (called a thumb) within a rectangle (called the slide). See the figure for an illustration. Within application defined limits, the user is free to move the thumb to a new position: the program can ask for the value of this new position at any time.

D.1 Using The Slide Potentiometer

All slide potentiometer routines begin with the letters 'sp_' (for slide potentiometer). All variables are integers, with the exception of window which is of the type WINDOW, slide which is of the type SLIDE_P, and orientation, which must be W_VERTICAL or W_W_HORIZONTAL. It is best to actually try out the demonstration program before you program with it. It is a fairly complex control which does its own event handling - put it through its paces!

Initialization.

Before use, the slide potentiometer must be initialized (*sp_init*). As part of the initialization procedure, the currently active window must be supplied, as well as the orientation of the potentiometer. Orientation is given as W_VERTICAL or W_HORIZONTAL. A pointer identifying the slide potentiometer instance is returned which must be used in all subsequent calls.

The slide must then be positioned within the window (*sp_position*). This is given in pixel co-ordinates of the diagonal defining the rectangle containing the slide. Once the slide is positioned, all activity will take place within that rectangle. This is the only time that the programmer must worry about pixel locations. It is the programmer's responsibility to ensure that this area will not be overwritten, and to repositioned the slide when the window changes size.

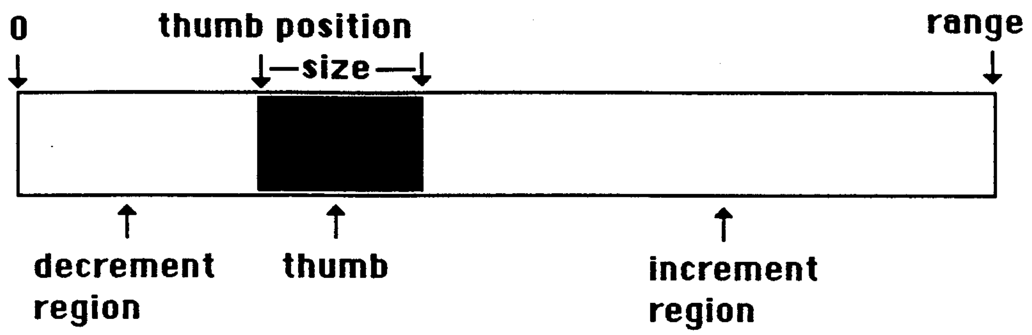
The applications programmer will also want to normalize the valuator to a world coordinate system (*sp_normalize*). In his co-ordinate system (not in pixels!) he is allowed to supply the range of values (from 0 to range), the position and size of the thumb, and the amount of increment or decrement that occurs when a point is pressed on either side of the thumb. Alternativly, the thumb can "snap" to the current point press rather than increment or decrement a given amount. If the bar is not normalized, it will assume defaults of a range of 0 - 100, thumb position of 0, thumb size of 10, with the "snap" on.

SLIDE_P sp_init (window, orientation)

> Initialize a Slide Potentiometer and return a pointer to it. This pointer must be used in all subsequent calls. NULL is returned on failure. *Orientation* is whether it should be treated vertically (W_VERTICAL) or horizontally (W_HORIZONTAL)

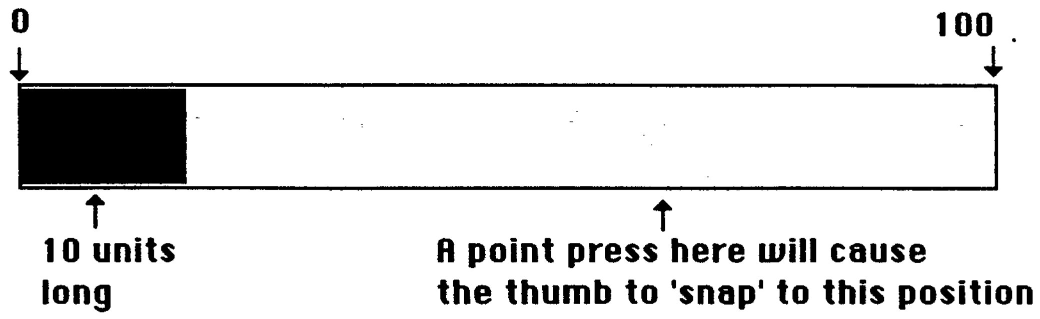
sp_position (slide, x1, y1, x2, y2)

The Slide Potentiometer



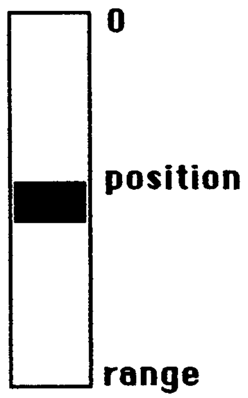
Example:

range = 100, position = 0, size = 10, inc/dec = W_SNAP


0
100
10 units long
A point press here will cause the thumb to 'snap' to this position

- After the thumb is moved by the user, the current position of the thumb will be returned as an integer between 0 and 100.

- In the vertical orientation, the top of the bar is 0 and the bottom is the range.



Position the slide onto the given *(x1,y1) (x2,y2)* pixel coordinates. The slide is automatically normalized to them, giving default values if necessary.

sp_normalize (slide, range, thumb_position, thumb_size, inc_dec_size)

Normalize the potentiometer to the given world co-ordinate system, i.e. if ***range*** is 100, values returned will be between 0-100. ***Thumb_position*** is the world position of the thumb and ***thumb_size*** is the relative world size of the thumb. ***Inc_dec_size*** is the amount of inc/decrement used in a point press on either side of the thumb. If set to the constant W_SNAP, the thumb will just move there. If a slide is not explicitly normalized, it defaults to a range of 100, position of 0, thumb size of 10, and W_SNAP.

Drawing.

Before anything is visible, one must draw the valuator (***sp_draw***). This activates the slide - until it is drawn, nothing will happen. This is usually done after initialization, window resizing, and if the application has somehow allowed the slide's area to be drawn over. In some applications (such as a scroll bar) the slide should be redrawn when the normalization is altered. There is no need to redraw the slide at any other time. The application may also 'hide' the slide potentiometer from view (***sp_erase***), which erases and inactivates it.

sp_draw (slide)

Draw the complete slide potentiometer This is only necessary initially or when the applications program has overwritten the area or re-normalized the slide. The slide is activated if it is not already active.

sp_erase (slide)

This erases the slide and inactivates it, ie a point press in bar area will return FALSE.

Active Use.

When a point event is detected, the programmer can detect if the point occured within the valuator (***sp_in_bar***) and if a user has completed an action in it. Three things may happen:

- A TRUE is returned if the user's initial point press was within the slide and if the action was completed.

- A W_CANCEL is returned if the initial point press was in the slide but the user cancelled his current action somehow (ie if you get a cancel, the point was in the slide but nothing meaningful was done with it). A W_CANCEL is also returned if the point was within the slide region but the slide was inactive.

- A FALSE is returned if the point was outside the slide.

The point is passed to the routine, and procedure does not return until the user has finished his current activity. All event handling is done internally - the programmer does not have to worry about repositioning or redrawing the thumb.

If a TRUE was returned, the current "value" can be obtained (***sp_get_units***). The application can then use this value in any way it likes.

int sp_in_bar (slide, x, y)

The workhorse. If the *(x,y)* point is within the slide, the thumb position will be automatically repositioned until the user stops and a TRUE is returned. If the point isn't within the slide, or if the user has somehow cancelled his operation, a FALSE is returned. Whatever happens, the thumb will always be in the correct position.

int sp_get_units (slide)

This function returns the current world coordinate position of the thumb. It should be used after a TRUE is returned from an sp_in_bar call.

Querying the Slide Potentiometer.

sp_get_window (slide)
Return the window the slide potentiometer is linked to.

Ending It All.

When the slide is no longer needed, the programmer can erase it from view (***sp_erase*** - discussed previously) and close it (***sp_end***).

sp_end (slide)

End the Slide Potentiometer. This frees up all the space

Useful defines.

W_VERTICAL	Orientation of the slide potentiometer. Used in 'orientation'
W_HORIZONTAL	Alternate orientation.
W_SNAP	The thumb 'snaps' to the point press. Used in ***inc_dec_size***

D.2 Example Program.

This section contains an example of how to use the facilities described earlier. The program ***slide.c*** is a process spawned by a driving program ***examples.c*** (see section B.4). Similar to cursor.c, it cannot be run independantly although the program modification to do so is simple.

This simple program will put up a slide potentiometer in the center of a window normalized to a range of 100, a thumb size of 10 and the initial thumb position at 0. Snap is initially on, although a pop-up menu item allows you to turn it off, with an increment/decrement of 10 units. A point press in the slide region will allow you to move the thumb. A "True" message is printed when a point press is in the slide, otherwise "False" is printed. Try changing the size of the window.

The executable process is called ***slide_vax*** and ***slide_cv*** for the vax and the corvus respectively. Compilation is similar to that described in section B.4. The sources and executable processes can be found in:

~ jade/examples/toolbox/Slide/slide.c
~ jade/examples/toolbox/Slide/slide_vax

```
~ jade/examples/toolbox/Slide/slide_ev
```

```
/*
 * ========================================================================
 * Source in ~jade/examples/toolbox/Slide/slide.c
 *
 * This is a CHILD process called from another program!
 * This  example program puts up a horizontal slide potentiometer
 * normalized to 100% (its default).
 * It reconstructs itself when the window is resized.
 * You may change the "snap" characteristics via a pop-up menu.
 * ========================================================================
 */

#include "slide.h"

main ()
{
    initialize ();                        /* Initialize the menu and      */
    while (TRUE) process_event ();        /* process events as they occur */
}

initialize () {
     j_process_id temp_wm;
     int temp_wi;

     j_initialize();                              /* Jipc stuff to create */
     j_send( j_parent_process());                 /* the child in a window*/
     j_reset();
     temp_wm = j_getp();
     temp_wi = j_geti();

     window = w_init (temp_wm, temp_wi);          /* Set up window & menu */
     w_layout_window(window, &w_layout);
     w_layout_menu(window, &menu1, items, NUM_MENU_ITEMS);
     w_buffered (window);

     slide = sp_init (window, 'H');               /* Set up the slide     */
     sp_normalize (slide, 100, 0, 10, W_SNAP);            /* Redundant as this is */
}                                                 /* also the default     */

process_event ()
{
     int      x, y, units;
     char     string[80];
     char     ignore_s[20];        /* A parameter we don't need           */
     int      ignore_int;          /* A parameter we don't need           */
     char     item_chosen[20];     /* The item on the menu selected       */


     switch (w_get_next_event (window)) {         /* Get the event        */

       /* CHANGE THE 'W_SNAP' CHARACTERISTICS OF THE POTENTIOMETER               */
       case WM_MENU_SELECTION:
         w_get_menu (window, &ignore_int, ignore_s, item_chosen, &ignore_int);

         if (strcmp (item_chosen, items[0].item_name) == 0)
              sp_normalize (slide, 100, sp_get_units (slide), 10, W_SNAP);

         else if (strcmp (item_chosen, items[1].item_name) == 0)
              sp_normalize (slide, 100, sp_get_units (slide), 10, 10);
         break;

       /* SIZE CHANGE? RELOCATE AND REDRAW CONTENTS                       */
       case WM_SIZE_SET:
         w_get_point (window, &x_size, &y_size);
         w_clear_rectangle_b (window, 0, 0, x_size, y_size);
         sp_position (slide, X1, Y, X2, Y + 20);
         sp_draw(slide);
         break;

       /* POINT PRESS?  GET THE POINT AND SEE IF ITS IN THE SLIDE        */
       case WM_POINT_DEPRESSED:  /* Point press.  Act on it                 */
         w_get_point (window, &x, &y);

         switch (sp_in_bar (slide, x, y)) {     /* Is point in the slide? */

           case TRUE:                          /* Yes, Something done     */
             w_put_characters_c(window, X2 + 5, Y + 20, "TRUE  ");
             units =  sp_get_units (slide);
             sprintf (string, "%3d%%  ", units);
             w_put_characters_c(window, X2 + 5, Y - 10, string);
             break;

           case FALSE:                         /* Not in the slide        */
             w_put_characters_c(window, X2 + 5, Y + 20, "FALSE ");
             break;

           case W_CANCEL:                              /* Yes, but cancelled      */
             w_put_characters_c(window, X2 + 5, Y + 20, "W_CANCEL");
         }
         break;

       /* USER DESTROYED WINDOW. GET OUT */
       case WM_DESTROY:
         exit (0);

     }
 }
```

```
/* Source in ~jade/examples/toolbox/Slide/slide.h                    */
#ifdef VAX
#       include <stdio.h>
#else
#       include <res_stdio.h>
#endif
#include <jipc/2.h>
#include <windows.h>
#include <tools.h>

#define X1       (x_size/3)      /* Slide will be in the middle third    */
#define X2       (x_size/3 * 2)  /* of the screen horizontally and half  */
#define Y        (y_size/2)      /* way up vertically                    */

WINDOW_LAYOUT w_layout = {       /* Window title and help screen         */
 "Slide Potentiometer",
 "Slide Potentiometer",
 "This demonstrates a slide potentiometer normalized to its defaults.\
  Try resizing it to see how it is rebuilt.\
  You may change the 'snap' characteristics through the pop-up menu.\
  Destroy the window to quit."
};

#define NUM_MENU_ITEMS  2

MENU_LAYOUT menu1 = {            /* Menu and item description            */
 "Slide",
 "This menu allows you to change the increment/decrement value."
 };

ITEM_LAYOUT items[NUM_MENU_ITEMS] = {
 "Snap on",
 "The thumb will 'snap' to the point press on either side of the thumb",
 W_IGNORE, W_IGNORE, W_IGNORE,

 "Snap off-10",
 "The thumb will increment/decrement 10 units on a point press on either\
  side of the thumb",
 W_IGNORE, W_IGNORE, W_IGNORE
};
                                        /* GLOBALS (Yuk)                 */
WINDOW  window;                         /* Window manager and window id */
SLIDE_P slide;                          /* The Slide potentiometer       */
int x_size, y_size;                     /* Current size of the window    */
```

E.

Tools Requiring Servers

Revision of 30 June 1985

E. Tools requiring servers.

The standard Jade window manager system is limited in its offerings. Although many functions can be extended by building higher level protocols (as in the toolbox desciption so far), there are some primitives which Jade does not supply. As these non-existing primitives are desirable in some applications, server programs must be downloaded to extend the powers currently available with the standard window manager.

Downloading a server is easy - just type the name of the server in an *other* window or type *cvcreate <server name>* in a vterm window. The server will exist locally as long as the corvus is not rebooted. All routines described here will fail (gracefully) if the server has not been downloaded.

E.1 Saving and restoring bitmaps.

Using standard Jade, it is currently impossible to save and restore a bitmap within a window. A special server, called *savemem*, overcomes this lack. It is downloaded by typing *savemem* in the *other* window or by typing *cvcreate savemem* in a vterm window. Once downloaded, use of following bitmap save and restore routines look like standard toolbox calls.

Saving a bitmap.

A bitmap may be saved upon request. The call will fail if savemem has not been downloaded or if there is not enough local memory to save the bitmap.

```
int w_save_bitmap (window, x1, y1, x2, y2)
    int x1, y1, x2, y2;
```

Save a bitmap defined by the absolute window coordinates *(x1,y1), (x2,y2)*. Return an integer which identifies the bitmap. This integer must be used during bitmap restoration. A FALSE is returned if the bitmap could not be saved.

Restoring a bitmap.

A bitmap that was previously saved may be restored or disposed of through the following calls. The variable *bitmap* is the integer returned by the *w_save_bitmap* call. Restoration of bitmaps may fail for a variety of reasons, such as window movement or resizing between saves and restores, or when the savemem server does not exist.

```
int w_restore_bitmap (window, bitmap)
    int bitmap;
```

Restore the identified *bitmap* to its previous location in the window. Return TRUE if everything went ok, False otherwise. The bitmap is discarded after it is restored.

```
int w_discard_bitmap (window, bitmap)
   int bitmap
```

Discard the identified *bitmap* without restoring it. This should always be used when you have saved a bitmap and do not wish to restore it. Neglecting to do so uses up memory on the workstation which may cause successive saves to fail. A TRUE or FALSE is returned denoting the call's success.

E.2 Example Program.

This section contains an example of how to use the bitmap routines. The program *bitmap.c* is a process spawned by a driving program *examples.c* (see section B.4). It cannot be run independantly although the program modification to do so is simple.

This simple program will look for a console window and save a pre-defined area within it. It will then put up a button over that area and write a message in it. After a short time, the bitmap will be restored and the console window will look as it did originaly.

To run, download the *savemem* process and open up a console window. Put something in the console window by sketching with the point button or through makeing a menu selecetion. Fire up the example program and go for the ride!

The executable process is called *bitmap_vax* and *bitmap_cv* for the vax and the corvus respectively. Compilation is similar to that described in section B.4. The sources and executable processes can be found in:

```
~jade/examples/toolbox/Bitmap/bitmap.c
~jade/examples/toolbox/Bitmap/bitmap_vax
~jade/examples/toolbox/Bitmap/bitmap_cv
```

```
/* Source in ~jade/examples/toolbox/Bitmap/bitmap.c
 * This is a CHILD process called from another program! */
#include <jipc/2.h>
#include <windows.h>
#include <tools.h>

WINDOW          window;
WINDOW          find_console ();
WINDOW_LAYOUT w_layout = {      /* Window title and help screen          */
 "Bitmap Demo",
 "Bitmap Demo",
 "This demonstrates how bitmaps can be saved and restored.\
  Make sure savemem is downloaded and open up a console window."
};

main ()         {
    j_process_id temp_wm;
    int temp_wi;

    j_initialize();                             /* Jipc stuff to create */
    j_send (j_parent_process() );               /* the child in a window*/
    j_reset();
    temp_wm = j_getp();
    temp_wi = j_geti();

    window = w_init (temp_wm, temp_wi);         /* Set up the window    */
    w_layout_window(window, &w_layout);
    w_clear_b (window);
    w_draw_string_b (window, 1,1,"Press point button in this window");
    w_draw_string_b (window, 2,1,"to get a message in the console window");
    w_draw_string_b (window, 3,1,"Make sure the console window is not blocked.");

    while (TRUE)  {                             /*  Process events      */
        switch ( w_get_next_event (window)) {
          case WM_POINT_DEPRESSED:
                message();
                break;
          case WM_DESTROY:
                exit (0);
          default:
                break;
        }
    }
}

message() {
    RECTANGLE   rectangle;
    WINDOW      console;
    char        message [100];
    int         bitmap;
    int         x_size, y_size, m, n ;

    if ( (console = find_console () ) == 0 ) {      /* Get console window */
         w_draw_string_b (window, 3,1,"Can't find the console window.    ");
         w_draw_string_b (window, 4,1,"Did you create it?                ");
         return;
    }

    w_get_window_size (console, &x_size, &y_size);  /* Save the bitmap   */
    bitmap = w_save_bitmap (console,  30, 30, x_size-30, y_size-30);
    if (bitmap == 0) {
         w_draw_string_b (window, 3,1,"Couldn't save map                 ");
         w_draw_string_b (window, 4,1,"Is savemem downloaded?            ");
         return;
    }

    strcpy (message, "Hey there cuty buns");        /* Put up the message */
    m = (x_size/2) - (strlen (message) * 3);
    n = y_size/2;
    rectangle = r_init (console);
    r_position (rectangle, 30, 30, x_size-30, y_size-30);
    r_outline (rectangle);
    r_erase (rectangle);
    w_put_characters_c (console, m, n, message);
    r_end (rectangle);
    sleep (6);

    if (w_restore_bitmap (console, bitmap) == 0) {  /* Destroy message  */
         w_draw_string_b (window, 3,1,"Couldn't restore map              ");
         w_draw_string_b (window, 4,1,"Was the window moved?");
    }
}

WINDOW find_console () {                        /* Find console window   */
    WINDOW console;
    char title[100];
    int window_list[20];
    int numb_windows, i;

    numb_windows = wm_get_window_list (w_get_manager (window), window_list, 20);
    for (i=0; i < numb_windows; i++) {
        console = w_init (w_get_manager(window), window_list[i]);
        w_get_title (console, title);
        if (strncmp (title, "Console", 6) == 0 )
            return (console);
        w_end (console);
    }
    return ( (WINDOW) 0);
}
```

Appendices.

Revision of 30 June. 1985

Appendix 1. Summary of Primitive Window Calls

Initializing and ending a window.

```
WINDOW w_init (window_manager, window_identifier)
w_end (window)
```

Requests of window status.

```
j_process_id w_get_manager (window)
int w_get_id (window)
int w_is_buffered (window)
int w_is_stroke_mode (window)
w_get_info (window, x1, y1, x2, y2, color)
```

Buffering output requests.

```
w_buffered (window)
w_unbuffered (window)
w_flush (window)
```

Events

WM_MENU_SELECTION	The user has made a menu selection.
WM_POINT_DEPRESSED	The point button was depressed.
WM_POINT_RELEASED	The point button was released.
WM_POINT_STROKE	The point button was held down while the mouse was moved.
WM_SIZE_SET	The user has changed The window size.
WM_KEY	The user has hit a key on the keyboard.
WM_CANCEL	The user has depressed a menu button and released it without making a menu selection. This is interpretted as a cancel.
WM_DESTROY	The user has destroyed the window

```
int w_any_events (window)
int w_get_next_event (window)
w_put_back_event (window)

w_get_menu (window, pane, menu, item, new_window)
w_get_point (window, x_coord, y_coord)
w_get_key (window, key)
```

Requests to the window manager.

```
w_set_title (window, window_title)
w_get_title (window, window_title)

w_set_pane_division (window, division_number, orientation, distance)
w_add_menu (window, division_vector, division_mask, menu_title, help)
w_add_simple_menu (window, menu_title, help)
w_delete_menu (window, menu_title)
w_make_window (window, menu_title, menu_item, x_default, y_default)
w_add_help (window, division_vector, division_mask, help)
w_delete_help (window, help_title)
w_add_simple_help (window, help)
w_item_help (window, menu_title, menu_item, help)

w_set_cursor (window, cursor_description, x_origin, y_origin)
w_get_cursor (window, cursor_description, x_origin, y_origin)

w_set_stroke_mode (window, stroke_mode)
w_get_window_size (window, x_size, y_size)
w_get_text_line (window, line_number, text_line, max_string_length)
```

Output to the window.

Routines with the parameter *color* have corresponding macro calls suffixed with '_b' '_c' which allow the *color* parameter to be dropped. Where sensible, the macro suffixed *_xor* may be used.

```
w_clear (window, color)

w_draw_string (window, line, column, color, string)
w_put_characters (window, x, y, color, string)

w_position_cursor (window, line, col)
w_erase_cursor (window, line, col)

w_scroll_vertically (window, line, number_of_lines, amount, color)
w_scroll_horizontally (window, line, col, number_of_cols, amount, color)
w_erase_lines (window, line, col, number_of_lines, color)

w_draw_vector (window, x1, y1, x2, y2, color)
w_put_bits (window, x, y, number_of_bits, color, bit_pattern)

w_clear_rectangle (window, x, y, x_extent, y_extent, color)
w_scroll_rectangle (window, x, y, x_extent, y_extent, direction, amount, color)
w_raster_copy (window, x_src, y_src, x_dest, y_dest, x_extent, y_extent)

w_ring_bell (window)
```

Program control of screens.

```
wm_get_screen_size (window_manager, x_size, y_size)
int wm_get_window_list (window_manager, list, max_windows)
int wm_create (window_manager, x1, y1, x2, y2)
int wm_init_w_indow_creation (window_manager, default_x_size, default_y_size)
wm_move_cursor (window_manager, x, y)
```

Program control of windows.

```
w_bury (window)
w_raise (window)
w_reverse (window)
w_destroy (window)
w_place (window, x1, y1, x2, y2)
```

Constants.

The following definitions are found in the include file *toolbox1.h*. Constants returned by w_get_next_event are described in the event section above.

W_BACKGROUND	Draw in background color
W_CONTRAST	Draw in reverse background color
W_XOR	Draw in exclusive or mode
WM_CH_HEIGHT	Height of a character in pixels
WM_CH_WIDTH	Width of a character in pixels
W_VERTICAL	Vertical pane orientation
W_HORIZONTAL	Horizontal pane orientation
W_IGNORE_PANE	Ignore panes in window
TRUE	True (1)
FALSE	False (0)
WM_UP	Scroll up
WM_DOWN	Scroll down
WM_LEFT	Scroll left
WM_RIGHT	Scroll right

Appendix 2. Cursor, Windows & Menu Summary

Changing the cursor.

```
w_change_cursor (window, cursor_type)
where cursor type is one of:
    DEFAULT_CS      LEFT_ARROW_CS  GLASSES_CS     PENCIL_CS
    HOURGLASS_CS    TARGET_CS      STASH_CS       GROUCHO_CS
    CORNER_LL_CS    CORNER_LR_CS   CORNER_UL_CS   CORNER_UR_CS
    X_CS            CROSS_CS       CONFIRM_CS
```

Setting up a window.

```
w_layout_window (window, window_appearance)
WINDOW_LAYOUT window_appearance;
where:
    typedef struct window_layout {          Window layout information:
        char title_bar [];                  -Title bar of window
        char window_name [];                -Name of help window
        char window_help [];                -Contents of help window
    } WINDOW_LAYOUT;
```

Setting up a menu.

```
w_layout_menu (window, menu_appearance, item_appearance, number_items)
w_layout_menu_in_pane (window, menu_appearance, item_appearance, number_items)
                        division_vector, division_mask)
MENU_LAYOUT menu_appearance;
ITEM_LAYOUT item_appearance[];

Where:
    typedef struct menu_layout {            Menu layout information
        char menu_name [];                  -Name of menu & help window
        char menu_help [];                  -Contents of help window
    } MENU_LAYOUT;

    typedef struct item_layout {            Item layout information
        char item_name [];                  -Name of item & help window
        char item_help [];                  -Contents of help window
        int     new_window_flag;            -Make a new window flag
        int     x_default_size;             -Default (x,y) window size
        int     y_default_size;              in pixels
    } ITEM_LAYOUT;

Useful constants:
    W_MAKE_WINDOW        W_DEFAULT_COLUMNS
    W_DEFAULT_LINES      W_IGNORE
```

Appendix 3. Rectangle Summary

Rectangle.

Initialization:

RECTANGLE r_init (window)
RECTANGLE r_copy (old_rectangle)
r_position (rectangle, x1, y1, x2, y2)
r_set_borders (rectangle, left_border, right_border, top_border, bottom_border)
r_set_all_borders (rect, border)
r_set_fill_margin (rectangle, fill_margin)

Drawing:

r_outline (rectangle)
r_fill (rectangle)
r_fill_all (rectangle)
r_fill_xor (rectangle)
r_fill_xor_all (rectangle)
r_erase (rectangle)
r_erase_all (rectangle)
int r_is_filled (rectangle)

Active use:

int r_in_rectangle (rectangle, x, y)
int r_within_rectangle (rectangle, x, y)

Querying:

WINDOW r_get_window (rectangle)
r_get_corners (rectangle, left, bottom, right, top)
int r_get_left (rectangle)
int r_get_right (rectangle)
int r_get_top (rectangle)
int r_get_bottom (rectangle)
r_get_borders (rectangle, left, bottom, right, top)
int r_get_left_border (rectangle)
int r_get_right_border (rectangle)
int r_get_top_border (rectangle)
int r_get_bottom_border (rectangle)
int r_get_fill_margin (rectangle)
int r_get_height (rectangle)
int r_get_width (rectangle)
int r_get_area (rectangle)

Ending it:

r_end (rectangle)

Appendix 4. Slide Potentiometer Summary

Initialization:

```
    SLIDE_P sp_init (window, orientation)
    sp_position (slide, x1, y1, x2, y2)
    sp_normalize (slide, range, thumb_position, thumb_size, inc_dec_size)
```

Drawing:

```
    sp_draw (slide)
    sp_erase (slide)
```

Active use:

```
    int sp_in_bar (slide. x, y)
    int sp_get_units (slide)
```

Querying:

```
    WINDOW sp_get_window (slide)
```

Ending it:

```
    sp_end (slide)
```

Constants:

```
    W_VERTICAL      W_HORIZONTAL      W_SNAP        W_CANCEL
```

Appendix 5. Tools Requiring Servers Summary

Saving and restoring bitmaps

These routines require the server *savemem* to be downloaded to the workstation.

```
int w_save_bitmap (window, x1, y1, x2, y2)
int w_restore_bitmap (window, bitmap)
int w_discard_bitmap (window, bitmap)
```

Appendix 6. Example Programs & Compiling

Compiling programs.

Programs running the toolbox must have the include file <*tools.h*> Programs normally are compiled for the vax by:

```
/usr/local/jade/bin/cc          your program here -ltools -lvterm -ljipc/2
```

Programs normally are compiled for the corvus by:

```
/usr/local/jade/bin/ccx          your program here -ltools -ljipc/2
```

Example programs.

All pathnames described here should be prefixed with ~ *jade/examples/toolbox/....* Pathnames on programs running locally do not allow the ~ convention. Additonally, local programs must be prefixed on vax C by */vaxc//user....* Programs running on the vax are invoked from a *vterm* window; on the corvus they are invoked in an *other* window.

Trirec/trirec_vax

Demonstrates menu options for drawing a triangle, rectangle, or quitting. Illustrates some toolbox1 primitives, such as menu and help window creation, event handling and simple output. Source in *trirec_vax.c.* Runs on the vax.

Trirec/trirec_cv

Similar to above, but runs on the corvus. Source in *trirec_cv.c.*

Manipulate/manipulate_windows

This program exists which demonstrates most of the window manipulation routines. To run it, security must be set to OFF (by selecting security from the OPTIONS menu in the console window). The program is intensionaly disconcerting to make you realize that program control of windows should not be abused. (No corvus version is present).

Trimessage/tri_message

Menu options draw a triangle in the current window or writting a message in a user-created window. Illustrates some toolbox2 primitives, such as menu and window construction, triggering new window creation on a given menu choice, and outputing to a window different from the current one. Source in *tri_message.c.* (No corvus version is present.)

Example_driver/examples_vax

Menu options spawn new (and independant) processes in user-created windows. By itself, it illustrates menu construction and use of *ifdefs* to make the same source code compilable for both the vax and the corvus. The programs spawned are discussed below. Source is in *examples.c*. *Note*: All the processes spawned are given by absolute pathnames. If you copy this program for test purposes, make sure these pathnames reflect your needs!

Example_driver/examples_cv

As above, but runs on the corvus.

Cursor/cursor_vax

This is a child process - it cannot be executed independantly. Menu options draw all cursors in the toolbox2 package. The *ChangeCursor* command is shown. Source is in *cursor.c*. Called from *examples_vax*..

Cursor/cursor_cv

As above, but runs on the corvus. Called from *examples_cv*.

Rectangle/rectangle_vax

This is a child process - it cannot be executed independantly. Illustrates use of some rectangle routines by using a rectangle as a toggle switch to print an off/on message. Source is in *rectangle.c*. Called from *examples_vax*.

Rectangle/rectangle_cv

As above, but runs on the corvus. Called from *examples_cv*.

Slide/slide_vax

This is a child process - it cannot be executed independantly. Illustrates use of most of the slide potentiometer routines by putting up a bar which prints out its current position plus what SPInBar returned. Snap may be turned on and off by a menu selection. Source is in *slide.c*. Called from *examples_vax*.

Slide/slide_cv

As above, but runs on the corvus. Called from *examples_cv*.

Bitmap/bitmap_vax

This is a child process - it cannot be executed independantly. Illustrates use of bitmap routines by saving and restoring bitmaps in the console window. Source is in *bitmap.c*. Called from *examples_vax*.

Bitmap/bitmap_cv

As above, but runs on the corvus. Called from *examples_cv*.